

Unit 1: Netlists

Abstract

This lecture covers some basic definitions, terminology, and results on netlists. Specifically, we will give a *syntax* and *semantics* to our designs; this will provide a formal model on which we can perform verification.

Definition 1 (Netlist) A netlist is an interconnection of *primitive circuit elements* (PCEs).

Definition 2 (PCE) Primitive circuit elements are the following:

1. Primary inputs
2. Gates: combinational functions (“Boolean”)
 - $f_G : \{0, 1\}^k \mapsto \{0, 1\}$ (k -ary gate)
 - list of k inputs (which are other nodes — notice the list is order-specific!)
3. Latches: memory units (variously “registers,” “delay elements,” etc.)
 - single input must be specified (“next-state”)

In addition —

1. Some of the elements will be designated as being *primary outputs* (POs)
2. Each node must have a unique “name” (variously “identifier,” “variable,” etc.)

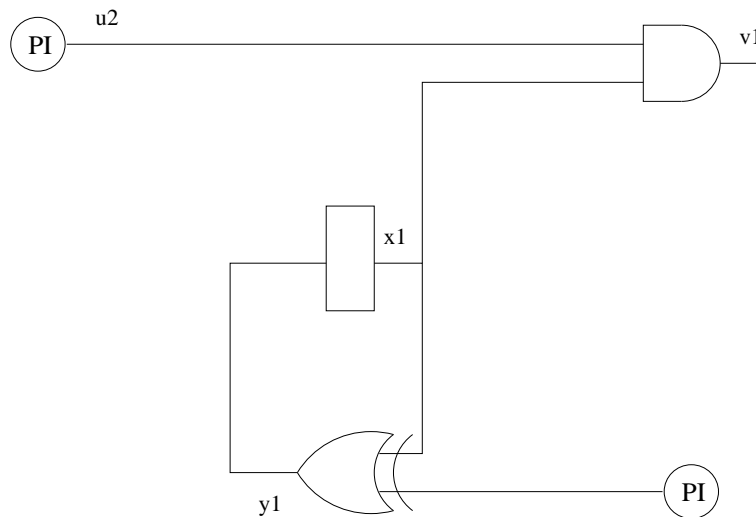


Figure 1: A netlist on one latch, two gates, two inputs, and one output.

Require: Netlist should have no “combinational cycles” (built into definition)

Example 1 (Netlist) See Figure 1.

Generalization: “Multivalued” netlist; operates on values from an arbitrary (finite) set.

- Same as before, except that gates are associated with function from $U_1 \times U_2 \cdots \times U_k \mapsto U_G$ where U_i, U_G are finite sets.

Example 2 (Multivalued Netlist) See Figure 2.

Informally, what we’ve done is give “syntax” to our designs. Now we will define a “semantics,” i.e., how they operate.

- Motivation: synchronous digital hardware, wherein all latches are driven by a single common clock, consequently changing state in lockstep. Referred to as “synchronous semantics”

Let’s start with purely combinational designs (i.e., no latches).

Example 3 (Combinational design) See Figure 3.

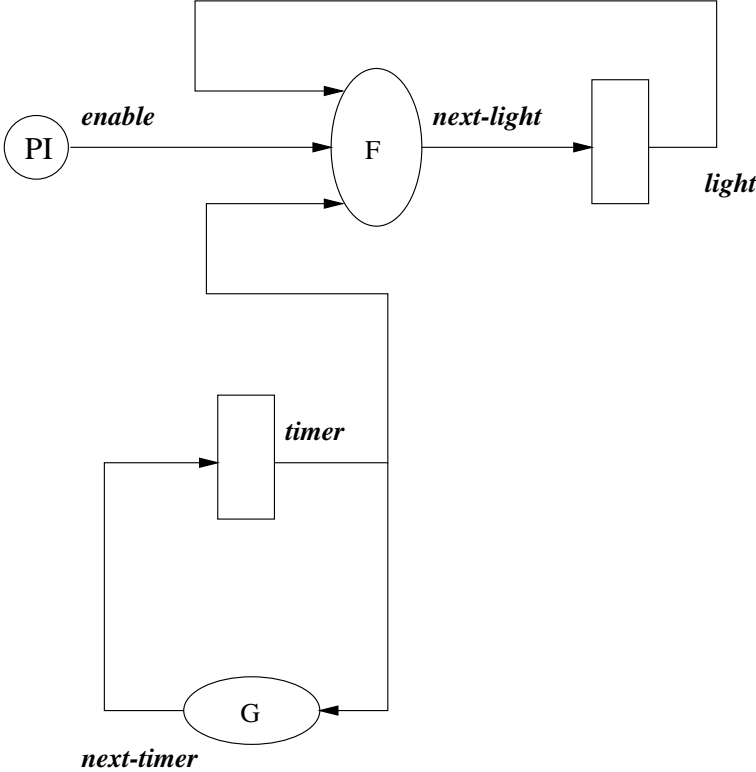


Figure 2: A multivalued netlist.

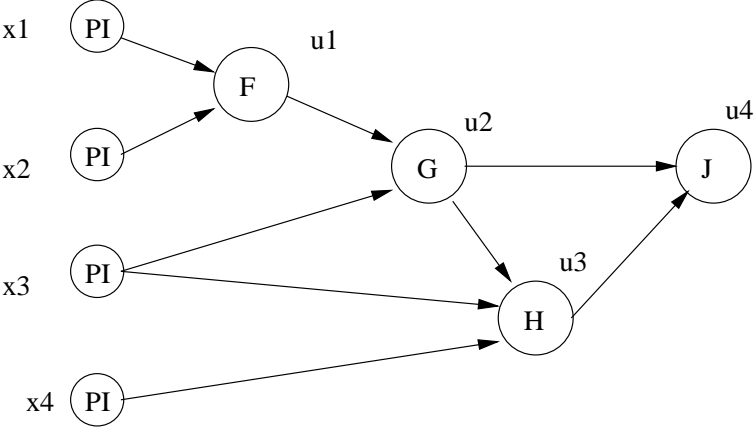


Figure 3: A combinational design.

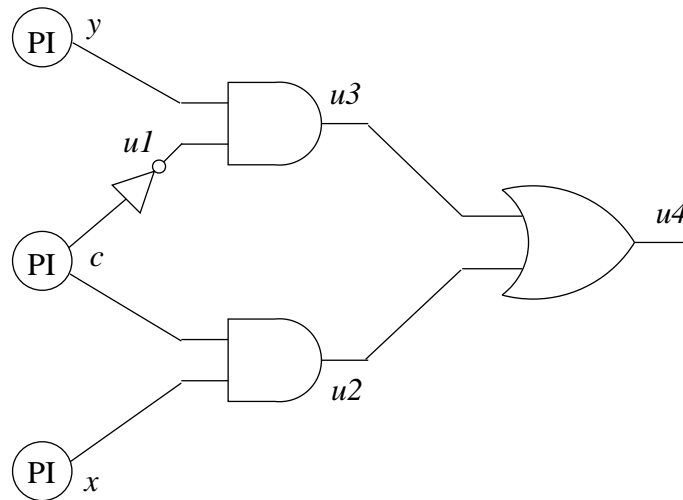


Figure 4: Functional composition

Observe: can naturally associate a Boolean function for each primary output in terms of the primary inputs using “functional composition.” E.g., for the design in Figure 3 we have:

$$\begin{aligned}
 u_4 &= J(u_2, u_3) \\
 &= J(G(u_1, x_3), H(u_2, x_3, x_4)) \\
 &= J(G(F(x_1, x_2), x_3), H(G(u_1, x_3), x_3, x_4)) \\
 &= J(G(F(x_1, x_2), x_3), H(G(F(x_1, x_2), x_3), x_3, x_4))
 \end{aligned}$$

Example 4 (Functional composition) See Figure 4.

Now we’ll give semantics to sequential designs.

Example 5 (Sequential design)

- Huffmann model of a sequential design — Figure 5.

First, we need to go over the basics of “sequences”.

Definition 3 (Sequence) Given a set (nonempty) \mathcal{A} , a (finite) sequence over \mathcal{A} is a function from $\{0, 1, 2, \dots, n - 1\}$ to \mathcal{A} for some $n \in \omega$, where $n > 0$. (The symbol ω denotes the set of natural numbers, i.e., $\{0, 1, 2, 3, \dots\}$.)

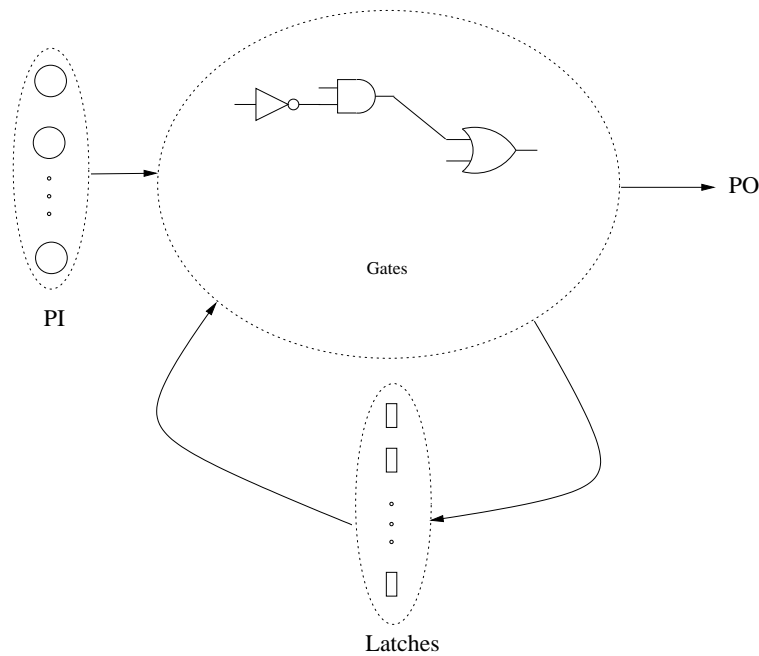


Figure 5: Sequential design — high-level view.

Example 6 (Sequences)

1. $A_1 = \{r, g, b\}$

- $f = \{(0, r), (1, g), (2, r), (3, r), (4, g)\}$.

2. $A_2 = \{0, 1\}^2 (= \{(0, 0), (0, 1), (1, 0), (1, 1)\})$

- $g(0) = (0, 0), g(1) = (0, 1), g(2) = (1, 0), g(3) = (1, 1)$

Notation: will find it convenient to denote a sequence $f : \{0, 1, 2, \dots, n - 1\} \mapsto A$ as follows:

- $\langle f(0), f(1), \dots, f(n - 1) \rangle$.

Terminology:

- the “length” of f is n
- can “concatenate” sequences in a natural way. Given $f : \{0, 1, 2, \dots, n - 1\}$ and $g : \{0, 1, \dots, m - 1\}$, then $h = f \frown g$ is the following function from $\{0, 1, \dots, n + m - 1\} \mapsto A \cup B$ — $h(k) = f(k)$ when $k < n$, and $h(k) = g(k - n)$ when $k \geq n$.

- can “project” a sequence on $A_1 \times A_2$. Given $f : \{0, 1, 2, \dots, n - 1\} \mapsto A_1 \times A_2$ then f projected to the first component is the sequence $g : \{0, 1, 2, \dots, n - 1\} \mapsto A_1$ where $g(k) =$ the first component of $f(k)$ (often denoted by $[f(k)]_1$).

Example 7 (Projection) Let $g : \{0, 1, 2, 3\} \mapsto \{0, 1\}^2$ be as previously defined. Then g projected to the first component is $\langle 0, 0, 1, 1 \rangle$.

Definition 4 (Language) A set of finite sequences over A will be referred to as a *language* over A .

Example 8 (Language) Take $A = \{0, 1\}$. Then the following are languages:

1. $L_1 = \{\langle 0 \rangle, \langle 0, 1 \rangle, \langle 0, 1, 0 \rangle, \langle 0, 1, 0, 1 \rangle\}$
2. $L_2 = \{f \mid f \text{ has an equal number of 0's and 1's}\}$
3. $L_3 = \{f \mid \text{every 0 in } f \text{ is followed by a 1}\}$

We are now ready to return to the original problem, namely that of giving semantics to sequential designs.

Definition 5 (Valuation) A *valuation* for a set of variables V is a function mapping V to $\{0, 1\}$.

- a valuation to the set of all PIs \equiv an “input”
- a valuation to the set of all latches \equiv a “state”
- a valuation to the set of all POs \equiv an “output”

Observe: given valuation to primary inputs and latches, can uniquely determine valuation for POs and next-state nodes by using functional composition. (Intuitively clear, can provide a mathematically rigorous proof of this fact.)

From this, we can infer the following critical fact: given a sequence of valuations to PIs, and a state, can uniquely derive a sequence of valuations to latches (“path”).

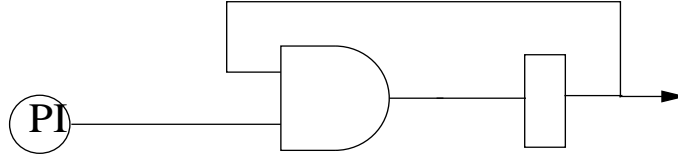


Figure 6: Deriving a path from an initial state, and an input sequence

- ditto for POs

Given p_0 — a state (i.e., a valuation to the latches) and a sequence of inputs $\langle i_0, i_1, \dots, i_{n-1} \rangle$:

Compute $(p_0, i_0) \mapsto p_1; (p_1, i_1) \mapsto p_2; (p_2, i_2) \mapsto p_3 \cdots (p_{n-1}, i_{n-1}) \mapsto p_n$

- path = $\langle p_0, p_1, \dots, p_n \rangle$

Example 9 (i/p seq + init state \mapsto path)

For POs: $p_0, \langle i_0, i_1, \dots, i_{n-1} \rangle$ as before. Then obtain $\langle p_0, p_1, \dots, p_n \rangle$, then $(p_0, i_0) \mapsto o_0, (p_1, i_1) \mapsto o_1, \dots, (p_{n-1}, i_{n-1}) \mapsto o_{n-1}$, and so obtain $\langle o_0, o_1, \dots, o_{n-1} \rangle$.

Let $\Sigma = I \times O$, where I = set of all inputs, and O = set of all outputs.

Observe: given initial evaluation to latches (i.e., an initial state) can naturally associate a language over Σ for netlist.

- Take all sequences $\langle (i_0, o_0), (i_1, o_1), \dots, (i_{n-1}, o_{n-1}) \rangle$ where $\langle i_0, i_1, \dots, i_{n-1} \rangle$ and p_0 yield $\langle o_0, o_1, \dots, o_{n-1} \rangle$.

Example 10 (Netlist \mapsto language)

- Figure 7: for the initial state 1 (call it s_0) L_{s_0} = all sequences of the form

$$- \langle (i_0, 0), (i_1, i_1), (i_2, 0), (i_3, i_3), \dots \rangle.$$

For example, $\langle (0, 0), (0, 0), (1, 0), (1, 1), (1, 0) \rangle \in L_{s_0}$.

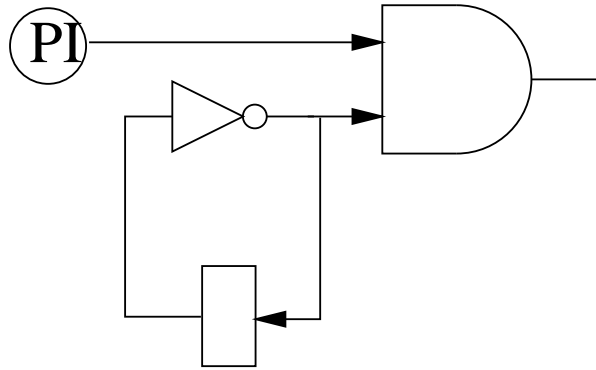


Figure 7: Deriving a language from a netlist

Equivalence

Fundamental question — when are two designs equivalent? (Ignore timing/area/testability — we’re only interested in functionality here!)

- Plug-In Plug-Out notion of equivalence — refer Figure 8

Assumption: “Pin correspondence” is given (i.e., PI/PO match up).

Given two netlist η_1 and η_2 , and initial state (s_0, t_0) in each

1. Suppose $L_{s_0}^{\eta_1} = L_{t_0}^{\eta_2}$. Is there any way these designs can be differentiated based on “observed” behavior?
2. Suppose $L_{s_0}^{\eta_1} \neq L_{t_0}^{\eta_2}$. Can these designs be differentiated on basis of observed behavior?
Let $\langle (i_0, o_0), (i_1, o_1), \dots, (i_{n-1}, o_{n-1}) \rangle \in L_{s_0}^{\eta_1}$ but not in $L_{t_0}^{\eta_2}$. Then on applying $\langle i_0, i_1, \dots, i_{n-1} \rangle$ as an input to η_1 and η_2 will result in different outputs — possibly faulty operation

Main issue: how to check equivalence? (Stay tuned — Unit 2!)

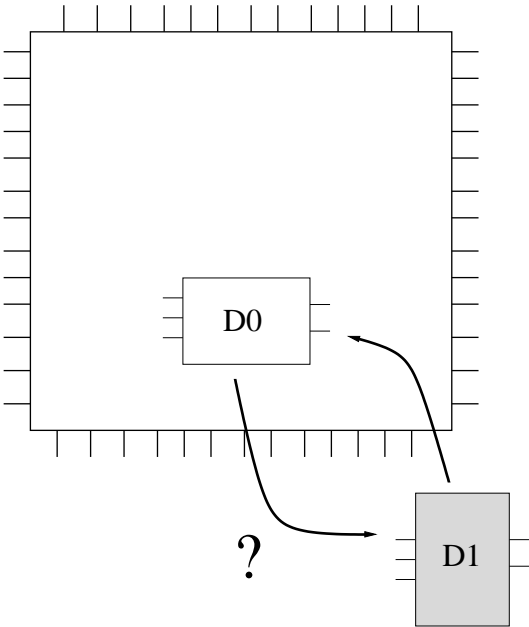


Figure 8: Plug-In Plug-Out equivalence